Avila Sofía Agustina 44208513

1) La opción C es correcto debido a que realiza correctamente lo esperado y utiliza los punteros de forma correcta.
En cambio el inciso A pasa por referencia el puntero y al avanzar en la lista, va cambiando su valor a l^.sig, cuando llegue al último nodo, el puntero será l:= nil y retornará ese puntero al programa, lo cual no es correcto.

El módulo B asigna de forma innecesaria aux:=l, directamente se podría recorrer con l ya que es pasado el puntero por valor.

El inciso D tampoco es correcto porque dentro del while, primero avanza en la lista y luego realiza la operación de duplicar por lo tanto, no todos los nodos tendrán su precio modificado. Lo que corresponde primero es duplicar su valor y luego avanzar al siguiente nodo.

2) Estoy de acuerdo con el alumno 1 y 2 porque de diferente forma cumplen lo esperado

No estoy de acuerdo con el alumno 3 porque no es correcto que una función no pueda retornar un puntero. Al ser un dato simple y definido por el programa, puede retornarlo.

Tampoco estoy de acuerdo con el alumno 4, porque un procedimiento puede retornar valores compuestos como propone el alumno 2.

Luego tendremos el puntero a ese empleado en el módulo desde donde fue invocado o en el programa principal, el cual de lo que ocurre en caso de que el proceso retorne el registro con la información del empleado

3) a) Falso. A pesar de que es correcto lo propuesto ya que en algún momento se encontrará el número buscado, no resultará eficiente. Lo recomendable será hacer una búsqueda dicotómica porque el arreglo se encuentra ordenado.
En este caso, al estar ordenado de mayor a menor se comparará el número buscado con el número del medio del vector y en caso que el número buscado sea menor que rc[medio], se continuará buscando en la segunda mitad del vector

~~b) Falso. Siempre es una opción posible obtener correcta modularización pero el problema que se presentaría será con las variables~~

b) Falso. La modularización siempre es correcto para tener un programa más legible pero el problema que habría con ello las variables globales será que podemos sobreescribir valores que otro módulo requiera y habrá confusión con el nombre de variables. Por eso, siempre es mejor las variables locales

c) Es válida porque la función recibe un único parámetro a pesar que realiza la operación A(20+y) · 20+y es posible porque ambos datos son de tipo entero

4)

```
const
    DimF = 1000;
```

4)

```
type
    vector = array [1..dimF] of integer;

    lista = ^nodo;
    nodo = record
      num: integer;
      sig: lista;
    end;
```

```
procedure cargarAlista (vc: vector; var l: lista; dimL: integer);
var nue: lista;
    i: integer;
begin l:= nil;
    for i:= 1 to dimL do begin
        new (nue);
        nue^.num := vc[i];
        nue^.sig := l;
        l:= nue;
    end;
end;
```

se espera que sea declarado en el programa principal y actualizado cuando se realicen las cargas de números

se actualizado su valor cuando se realizó la carga de números

```
var vc: vector;
begin
```

```
procedure cargarAlista (vc: vector; var l: lista; dimL: integer);
var i: integer;
begin
    for i:= 1 to dimL do begin
```

5)

Antes de For 1. Estática = 4 + 400 = 404 bytes 408 bytes
Después de For 1 Estática = 408 bytes 800
Dinámica: 800 bytes = 1208 bytes

Antes de for 2. Estática: 408 bytes 800
Dinámica: 1208 bytes
Después for 2 Estática: 408 bytes
Dinámica: 800 bytes

Después for Después for 3 Estática: 408 bytes
Dinámica: 800 - 400 - 400 bytes.